Improving the Performance of Discrete Lagrange-Multiplier Search for Solving Hard SAT Problems

Yi Shang*

Dept. of Computer Engineering/Science
University of Missouri at Columbia
Columbia, MO 65211, USA
http://www.cecs.missouri.edu/

Benjamin W. Wah†

Dept. of Electrical and Computer Engineering
and the Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
Urbana, IL 61801, USA
http://manip.crhc.uiuc.edu

Abstract

Recently, we have proposed the discrete Lagrange-multiplier method (DLM) to solve satisfiability problems. Instead of restarting from a new starting point when the search reaches a local minimum in the objective space, the Lagrange multipliers of violated constraints in DLM provide a force to lead the search out of the local minimum and move it in a direction provided by the multipliers. In this paper, we present the theoretical foundation of DLM for solving SAT problems and discuss some implementation issues. We study the performance of DLM on a set of hard satisfiability benchmark instances, and show the importance of dynamic scaling of Lagrange multipliers and the flat-move strategy. We show that DLM can perform better than competing local-search methods when its parameters are selected properly.

1 Introduction

Satisfiability (SAT) problems are a class of discrete constraint-satisfaction problems (CSP) that model a wide range of real-world applications. They are NP-complete and require algorithms of exponential complexity in the worst case to obtain a solution.

A SAT problem is defined as follows. Given a set of n clauses $\{C_1, C_2, \cdots, C_n\}$ on m variables $\mathbf{x} = (x_1, x_2, \cdots, x_m)$, $x_i \in \{0, 1\}$, and a Boolean formula in conjunctive normal form (CNF),

$$C_1 \wedge C_2 \wedge \cdots \wedge C_n, \qquad (1)$$

find an assignment to $\mathbf{x}$ so that (1) evaluates to be *true*, or derive its infeasibility if (1) is infeasible.

Many search methods have been developed in the past for solving SAT problems. These can be classified as incomplete and complete, depending on whether they can find a random solution or find all the solutions. Complete algorithms include resolution [1, 6], backtracking, consistency testing [2], and integer programming methods [3]. These methods enumerate the search space systematically. Their advantage is that they can detect infeasibility when a SAT problem is infeasible, but are generally computationally expensive and have difficulties in solving large instances.

On the other hand, incomplete methods are much faster and can solve some hard SAT problems with size of an order-of-magnitude larger than those solved by complete methods. Many incomplete local-search methods have been designed to solve large SAT problems of thousands of variables [2, 7, 9]. However, they may be trapped by local minima in the objective space without satisfying all the constraints. Global-search strategies have been introduced to bring a search out of a local minimum in the objective space. These include multi-start (restart) of descent methods, and stochastic methods, such as simulated annealing (SA) and genetic algorithms (GA). The disadvantage of random restarts is that they may discard their search history and simply bring the search to a completely new region, whereas stochastic methods, such as GA and SA, are computationally expensive and do not handle constraints well.

Recently, we have developed an efficient incomplete search method to solve SAT problems based on a dis-

*Research supported by Research Board Grant of University of Missouri System.

†Research supported by National Science Foundation Grant MIP 96-32316. Currently on leave at the Department of Computer Science and Engineering, Chinese University of Hong Kong, Shatin, Hong Kong.

Proceedings of 10th International Conference on Tools with Artificial Intelligence, IEEE, Nov. 1998.

crete Lagrangian formulation [9, 10]. We proved first-order conditions (similar to those in continuous space) for a Lagrangian formulation of discrete optimization problems, and developed first-order search procedures (DLM) to find points satisfying the discrete-space first-order conditions. Instead of restarting from a new starting point when a search reaches a local trap in the objective space, the Lagrange multipliers in DLM provide a force to lead the search out of the local minimum and move it in the direction provided by the Lagrange multipliers. Hence, DLM does not restart itself and avoids breaks in the trajectory as in restart methods. This is advantageous when the trajectory is already in the vicinity of a satisfiable solution, and a random restart may bring the search to a totally different search space. Moreover, DLM has very few algorithmic parameters to be tuned, and the procedure can be made deterministic and the results, reproducible.

When applied to solve DIMACS SAT benchmarks, DLM generally performs better than the best competing methods and can achieve an order-of-magnitude speedup for many problems. However, DLM is still eluded by some hard instances. In this paper, we take a closer look at the parameters of DLM and investigate their effects in solving hard SAT problems. Following the approach in [9], we formulate a SAT problem as a discrete constrained optimization problem as follows:

$$\begin{array}{ll} \min_{\mathbf{x}\in\{0,1\}^m} & N(\mathbf{x}) = \sum_{i=1}^{n} w_i U_i(\mathbf{x}) \\ \text{subject to} & U_i(\mathbf{x}) = 0 \quad \forall i \in \{1,2,\ldots,n\}, \end{array} \tag{2}$$

where w_i is a positive integer constant, and $U_i(\mathbf{x})$ equals 0 if $\mathbf{x}$ satisfies C_i, and 1 otherwise.

This paper is organized as follows. Section 2 presents DLM, followed by a discussion of issues and alternatives in implementing DLM in Section 3. Section 4 presents experimental results in applying DLM to solve a set of hard test problems. Several important aspects of DLM are studied by examining the corresponding algorithmic parameters.

2 Background

In this section we first summarize previous work on Lagrangian methods for solving continuous constrained optimization problems. We then extend continuous Lagrangian methods to solve discrete problems and apply them to solve SAT problems.

2.1 Lagrangian Methods

Lagrangian methods are classical methods for solving continuous constrained optimization problems [4]. An equality constrained optimization problem is defined as follows:

$$\begin{array}{ll} \min_{\mathbf{x}\in E^m} & f(\mathbf{x}) \\ \text{subject to} & g(\mathbf{x}) = 0 \end{array} \tag{3}$$

where $\mathbf{x} = (x_1, x_2, \cdots, x_n)$ are real variables, and $g(\mathbf{x}) = (g_1(\mathbf{x}), g_2(\mathbf{x}), \cdots, g_n(\mathbf{x}))$ are n constraints. Lagrangian function L_c is defined as follows:

$$L_c(\mathbf{x}, \lambda) = f(\mathbf{x}) + \sum_{i=1}^{n} \lambda_i g_i(\mathbf{x}) \tag{4}$$

where $\lambda = (\lambda_1, \cdots, \lambda_n)$ are Lagrange multipliers.

A *saddle-point* $(\mathbf{x}^*, \lambda^*)$ of Lagrangian function $L_c(\mathbf{x}, \lambda)$ is defined as one that satisfies the following condition.

$$L_c(\mathbf{x}^*, \lambda) \le L_c(\mathbf{x}^*, \lambda^*) \le L_c(\mathbf{x}, \lambda^*) \tag{5}$$

for all $(\mathbf{x}^*, \lambda)$ and all $(\mathbf{x}, \lambda^*)$ sufficiently close to $(\mathbf{x}^*, \lambda^*)$.

A point x satisfying constraints $g(\mathbf{x}) = 0$ is said to be a *regular point* [4] of the constraints if gradient vectors $\nabla h_1(x), \cdots, \nabla h_m(x)$ are linearly independent.

(Continuous) First-order necessary conditions [4]. Let x be a local extremum point of $f(x)$ subject to constraints $g(x) = 0$. Further, assume that $x = (x_1, \ldots, x_n)$ is a regular point of these constraints. Then there exists $\lambda \in E^m$ such that

$$\nabla_x f(x) + \lambda^T \nabla_x g(x) = 0 \tag{6}$$

Based on the definition of Lagrangian function, the necessary conditions can be expressed as follows:

$$\begin{aligned} \nabla_x L(x, \lambda) &= 0, \\ \nabla_\lambda L(x, \lambda) &= 0. \end{aligned} \tag{7}$$

Using the first-order necessary and second-order sufficient conditions (not shown), numerical algorithms have been developed to look for points satisfying these conditions. One typical method, called the *first-order method*, does descents in the original variable space of $\mathbf{x}$ and ascents in the Lagrange-multiplier space of λ. The method can be written as a set of ordinary differential equations as follows:

$$\frac{d\mathbf{x}}{dt} = -\nabla_{\mathbf{x}} L_c(\mathbf{x}, \lambda) \quad \text{and} \quad \frac{d\lambda}{dt} = \nabla_\lambda L_c(\mathbf{x}, \lambda) \tag{8}$$

where t is an autonomous variable and ∇ is the gradient operator.

To apply Lagrangian methods to solve discrete optimization problems, we need to develop the counterpart of gradient in discrete space. Consider the following equality-constrained optimization problem in discrete space, which is similar to the continuous version in (3),

$$\begin{array}{ll} \min_{\mathbf{x}\in D^m} & f(\mathbf{x}) \\ \text{subject to} & g(\mathbf{x}) = 0 \end{array} \tag{9}$$

where D is a subset of integers.

Definition. A *local minimum* $\mathbf{x}^*$ to (9) subject to constraints is defined as $g(\mathbf{x}^*) = 0$ and $f(\mathbf{x}^*) \leq f(\mathbf{x})$ for any feasible $\mathbf{x}$, and $\mathbf{x}^*$ and $\mathbf{x}$ differ in only 1 dimension by a magnitude of 1.

A local minimum is defined with respect to a certain *neighborhood.* In this definition, the neighborhood is within 1 of one dimension. Note that this definition can be extended to the case in which two points differ by more than one dimensions [10].

Discrete Lagrangian function L_d has the same form as in the continuous case in (4). A saddle point $(\mathbf{x}^*, \lambda^*)$ of $L_d(\mathbf{x}, \lambda)$ is reached when the following condition is satisfied:

$$L_d(\mathbf{x}^*, \lambda) \leq L_d(\mathbf{x}^*, \lambda^*) \leq L_d(\mathbf{x}, \lambda^*) \tag{10}$$

for all λ sufficiently close to λ^* and all $\mathbf{x}$ that differ from $\mathbf{x}^*$ in only one dimension by a magnitude 1.

In a way similar to the continuous case, the following theorem states the discrete-space first order conditions.

(Discrete) First-Order Necessary Conditions [10]. In discrete space, the set of all saddle points is equal to the set of all solutions that satisfy the following two equations:

$$\begin{aligned} \Delta_x L_d(x, \lambda) &= \Delta_x[f(x) + \lambda^T g(x)] = 0 && (11) \\ \nabla_\lambda L_d(x, \lambda) &= 0 && (12) \end{aligned}$$

Note that the gradient operator in (11) is discrete, that the discrete gradient operator *cannot* be distributed into the two terms of the addition, and that the gradient in (12) can be continuous.

In the continuous case, methods that look for saddle points utilize gradient information. In order for these methods to work in discrete variable space $\mathbf{x}$, we define the discrete descent operator $\Delta_\mathbf{x}$ as follows:

Definition. Discrete descent operator $\Delta_\mathbf{x}$ is defined with respect to $\mathbf{x}$ in such a way that $\Delta_\mathbf{x} L_d(\mathbf{x}, \lambda) = (\delta_1, \cdots, \delta_m) \in \{-1, 0, 1\}^m$, $\sum_{i=1}^m |\delta_i| = 1$, $(\mathbf{x} - \Delta_\mathbf{x} L_d(\mathbf{x}, \lambda)) \in D^m$, and

$$L_d(\mathbf{x} - \Delta_\mathbf{x} L_d(\mathbf{x}, \lambda), \lambda) \leq L_d(\mathbf{x}, \lambda).$$

Further, if $\forall \mathbf{x}'$ such that $\sum_{i=1}^m |x_i' - x_i| = 1$ and $L_d(\mathbf{x}, \lambda) \leq L_d(\mathbf{x}', \lambda)$, then $\Delta_\mathbf{x} L_d(\mathbf{x}, \lambda) = 0$.

Based on this definition, first-order methods for continuous problems can be extended to work for discrete problems. The basic idea is to descend in the original discrete variable space of $\mathbf{x}$ and ascend in the Lagrange-multiplier space of λ.

Generic First Order Discrete Lagrangian Method (DLM)

$$\begin{aligned} \mathbf{x}^{k+1} &= \mathbf{x}^k - \Delta_\mathbf{x}(\mathbf{x}^k, \lambda^k) && (13) \\ \lambda^{k+1} &= \lambda^k + g(\mathbf{x}^k) && (14) \end{aligned}$$

where $\Delta_\mathbf{x}(\mathbf{x}^k, \lambda^k)$ is the discrete descent operator for updating $\mathbf{x}$ based on Lagrangian function $L_d(\mathbf{x}, \lambda)$.

DLM provides a theoretical foundation and generalization of local-search schemes that optimize the objective alone and clause-weight schemes that optimize the constraints alone. In contrast to local-search methods that restart from a new starting point when a search reaches a local trap in the objective space but not a satisfiable solution, Lagrange multipliers in DLM provide a force to lead the search out of the local minimum and move it in the direction provided by the Lagrange multipliers.

In contrast to constraint-weight schemes that updates weights dynamically based on which constraints are violated after each flip or after a series of flips, DLM also uses the value of the objective function to provide further guidance. The value of the objective function may be important in some stages of a search. For instance, when the number of violated constraints is large, it is important to reduce the number of violated constraints by choosing proper variables to flip. In this case, the weights on the violated constraints may be relatively small as compared to the number of violated constraints and, hence, play a minor role in determining the variables to flip. The dynamic shift in emphasis between the objective and the constraints, depending on their relative values, is the key of DLM.

2.2 DLM Formulation of SAT

To apply DLM to solve a SAT problem, we introduce an artificial objective $H(\mathbf{x})$ and formulate the problem into a constrained optimization problem as follows:

$$\begin{array}{lll} \min_{\mathbf{x}\in\{0,1\}^m} & H(\mathbf{x}) & \\ \text{subject to} & U_i(\mathbf{x}) = 0 & \forall i \in \{1, 2, \ldots, n\}, \end{array} \tag{15}$$

where $U_i = 0$ if C_i is satisfied, and $U_i = 1$ if C_i is not satisfied. DLM searches for a saddle point of the Lagrangian formulation of (15) defined as follows:

$$L(\mathbf{x}, \lambda) = H(\mathbf{x}) + \lambda^T U(\mathbf{x}), \tag{16}$$

where λ is a vector of Lagrange multipliers.

A saddle point $(\mathbf{x}^*, \lambda^*)$ to the Lagrangian formulation of (15) consists of a feasible solution $\mathbf{x}^*$ to the SAT problem in (1). Given the saddle point $(\mathbf{x}^*, \lambda^*)$,

$$L_d(\mathbf{x}^*, \lambda) \leq L_d(\mathbf{x}^*, \lambda^*) \Rightarrow \lambda^T U(\mathbf{x}^*) \leq \lambda^{*T} U(\mathbf{x}^*)$$

for any λ close to λ^*. This condition holds only when $U(\mathbf{x}^*) = 0$.

Many functions can be used as $H(\mathbf{x})$ such that saddle points of (15) contain feasible solutions to the SAT problem in (1). Examples of $H(\mathbf{x})$ are as follows:

$$H(\mathbf{x}) = \begin{cases} \sum_{i=1}^{n} l_i U_i(\mathbf{x}), & \text{or} \\ \sum_{i=1}^{n} (l_{max} + 1 - l_i) U_i(\mathbf{x}), & \text{or} \\ \sum_{i=1}^{n} w_i U_i(\mathbf{x}), \end{cases}$$

where l_i is the number of variables appearing in clause C_i, $l_{max} = \max_{i=1}^{n} l_i$, and w_i is some positive constant. The first function assigns more weights to longer clauses; the second one is the opposite; and the third assigns user-specified weights to all clauses. In this paper, we use the third function as our objective function and formulate SAT problems as constrained optimization problems defined in (2).

The SAT problem defined in (2) is a special case of the discrete constrained optimization problem defined in (9). An important property of the formulation in (2) is that all local minima are also global minima. This is true because, based on (2), $\mathbf{x}^*$ is defined as a local minimum if $U(\mathbf{x}^*) = 0$, which implies $N(\mathbf{x}^*) = 0$ (a global minimum).

Using the SAT formulation in (2), the Lagrangian function is

$$L(\mathbf{x}, \lambda) = N(\mathbf{x}) + \lambda^T U(\mathbf{x}) = \sum_{i=1}^{n} (w_i + \lambda_i) U_i(\mathbf{x}), \tag{17}$$

where $\mathbf{x} \in \{0,1\}^m$, $U(\mathbf{x}) \in \{0,1\}^n$, and $w_i, i = 1, \cdots, n$ are positive integer coefficients.

To apply DLM to solve SAT problems, we define the *discrete gradient operator* $\Delta_{\mathbf{x}} L(\mathbf{x}, \lambda)$ with respect to $\mathbf{x}$ such that $\Delta_{\mathbf{x}} L(\mathbf{x}, \lambda)$ points to state $\mathbf{x}'$ in the Hamming distance-1 neighborhood of the current state $\mathbf{x}$ that gives the maximum improvement in $L(\mathbf{x}, \lambda)$. If no state in the 1-neighborhood of $\mathbf{x}$ improves $L(\mathbf{x}, \lambda)$, then $\Delta_{\mathbf{x}} L(\mathbf{x}, \lambda) = 0$.

There are three important properties about DLM.

```
1. Set initial $\mathbf{x}$
2. Set initial $\lambda$
3. **while** $\mathbf{x}$ is not a feasible solution, *i.e.*, $N(\mathbf{x}) > 0$
4.     update $\mathbf{x}$: $\mathbf{x} \longleftarrow \mathbf{x} \oplus \Delta_{\mathbf{x}} L(\mathbf{x}, \lambda)$, $\oplus$ is XOR.
5.     **if** condition for updating $\lambda$ is satisfied **then**
6.         update $\lambda$: $\lambda \longleftarrow \lambda + c \times U(\mathbf{x})$
7.     **end if**
8. **end while**
```

Figure 1: Generic DLM $\mathcal{A}_1$ for solving SAT problems.

- Since $U(\mathbf{x}) = 0$ when $\mathcal{A}$ converges to a discrete saddle point, $\mathbf{x}$ is globally (not just locally) optimal when $\mathcal{A}$ converges. If any of the constraints in $U(\mathbf{x})$ is not satisfied, then λ will continue to evolve to handle the unsatisfied constraints, and the search continues.

- Since $U_i(\mathbf{x})$ is weighted by $w_i + \lambda_i$ in (17), changing λ can help the local search escape from local minima. The dynamic scaling of λ can provide the necessary shift of attention between the objective and the constraints to speed up the search.

- Since the objective and the constraints in (2) are dependent, the minimization of (17) is equivalent to a clause-weight scheme. This is different from the general case in which the objective and the constraints are independent. In the general case, descents in the original-variable space and ascents in the Lagrange-multiplier space form two counteracting forces to converge to saddle points.

3 Implementation Issues

In this section, we discuss various considerations in implementing DLM. Figure 1 shows the pseudo code of $\mathcal{A}_1$, a generic DLM implementing (13) and (14). Define one *iteration* as one pass through the *while* loop. In the following, we describe some of our design considerations.

(a) *Initial points* (Lines 1-2). DLM is started from either the origin or from a random initial point generated using a fixed random seed. Further, λ is always set to zero. Since DLM is deterministic with no randomness involved, the fixed initial points allow the results to be reproduced easily.

(b) *Descent and ascent strategies* (Line 4). There are two ways to calculate $\Delta_{\mathbf{x}} L(\mathbf{x}, \lambda)$: greedy and hill-climbing, each involving a search in the range of Hamming distance 1 from the current $\mathbf{x}$. In a *greedy strategy*, the assignment leading to the maximum decrease in $L(\mathbf{x}, \lambda)$ is selected to update the current assignment, whereas in *hill-climbing*, the first assignment leading to a decrease in $L(\mathbf{x}, \lambda)$ is selected. Depending on the order of search and the number of assignments that can be improved, hill-climbing strategies are generally much faster than greedy strategies.

Among various ways of hill-climbing, we used two alternatives in our implementation: flip the variables one by one in a predefined order, or maintain a first-in-first-out (FIFO) list of variables that can improve the current $L(\mathbf{x}, \lambda)$ and just flip the first variable in the list. The first alternative is fast when the search starts. By starting from a randomly generated initial assignment, it usually takes very few flips to find a variable that improves the current $L(\mathbf{x}, \lambda)$. As the search progresses, there are fewer variables that can improve $L(\mathbf{x}, \lambda)$. At this point, the second alternative becomes more efficient and should be applied.

(c) *Conditions for updating* λ (Line 5). The frequency in which λ is updated affects the performance of a search. The considerations here are different from those of continuous problems. In a discrete problem, descents based on discrete gradients usually make small changes in $L(\mathbf{x}, \lambda)$ in each update of $\mathbf{x}$ because only one variable changes. Hence, λ should not be updated in each iteration of the search to avoid biasing the search in the Lagrange-multiplier space of λ over the original variable space of $\mathbf{x}$.

Experimental results show that a good strategy is to update λ only when $\Delta_{\mathbf{x}} L(\mathbf{x}, \lambda) = 0$. At this point, a local minimum or a plateau is reached. The search updates λ immediately and keeps it unchanged for a while through the *flat-move* strategy, since updating λ in a plateau changes the surface of the plateau and may make it more difficult for the search to find a local minimum somewhere inside the plateau. The flat-move strategy allows the search to continue for some time in the plateau without changing λ. To prevent the search from being trapped in a plateau, we keep a tabu list consisting of variables flipped in the recent past and do not consider these variables in flat moves. We also set a limit on how many flat moves can be done in a plateau.

(d) *Amount of update of* λ (Line 6). A parameter c controls the magnitude of changes in λ. In general, c can be a vector of real numbers, allowing non-uniform updates of λ across different dimensions and possibly across time. In our experiments, $c = 1$ has been found to work well for most of the benchmarks tested. However, for some larger problems, a larger c could result in shorter search time and better solutions.

The update rule in Line 6 results in nondecreasing λ because $U(\mathbf{x})$ is either 0 or 1. In contrast, in continuous problems, λ_i of constraint $g_i(\mathbf{x}) = 0$ increases when $g_i(\mathbf{x}) > 0$ and decreases when $g_i(\mathbf{x}) < 0$.

From previous experience in solving difficult SAT problems using DLM, some λ values can become very large after tens of thousands of iterations. At this point, the Lagrangian search space (17) becomes very rugged, and the search has difficulty in identifying an appropriate direction to move. To cope with this problem, λ should be kept within a limited range. One way to achieve this is to reduce λ and w periodically. For instance, we found empirically that λ could be scaled down by a factor of 2 every 500 iterations in order to maintain a reasonable search space. This strategy reduces $L(\mathbf{x}, \lambda)$ and restricts the growth of Lagrange multipliers, leading to faster solutions for many test problems.

(e) *Integer operations for efficiency.* Since integer operations are much faster than floating-point operations, DLM was implemented using integer operations as much as possible. Variables $\mathbf{x}$, λ, and w are all integers. Scaling down of λ_i is done by dividing λ_i by a scale factor and rounding the number to the nearest integer.

Figure 2 shows $\mathcal{A}_3$, our implementation of DLM to solve SAT. $\mathcal{A}_3$ uses the first strategy of hill-climbing in descents in the beginning and switches to the second strategy later. We use κ to control the switch from the first strategy to the second. We found that $\kappa = n/3$ works well and used it in our experiments. $\mathcal{A}_3$ updates λ only when a local minimum or a plateau is reached. In our experiments, we have used a simple scheme that increases λ by a constant across all clauses. $\mathcal{A}_3$ controls the magnitude of $L(\mathbf{x}, \lambda)$ by periodically reducing λ. More sophisticated adaptive mechanisms can be developed to lead to better performance.

4 Experimental Results

DLM's performance has been demonstrated in solving an extensive set of benchmark probelms archived in DIMACS of Rutgers University [9]. DLM can solve most of these problems within seconds but cannot solve a few hard problems, including "*par16*" and "*par32*," even after a large number of iterations and long computation time. Note that "*par32*" defeats all current local-search methods and is listed as one of the challenges in propositional reasoning and search [8].

In this section, we focus on improving DLM on "*par16*" and "*par32*." Each problem has a compressed version and an uncompressed one. We only show our results on the compressed versions because each uncompressed version can be simplified into the corresponding compressed one. Written in C, DLM was

```
Set initial $\mathbf{x}$
Set $\lambda = 0$ and $c = 1$
Set the same constant weight $w_i$ for each clause, e.g., 2
Set $\kappa = n/3$, where $n$ is the number of variables
Set $\lambda$ reduction interval $I_\lambda$ (e.g. 1000)
Set reduction ratio $r$ (e.g. 1.5)
Set flat-move limit $L_f$, e.g., 1
Set tabu length $L_t$, e.g., 1
while $\mathbf{x}$ is not a feasible solution, i.e., $N(\mathbf{x}) > 0$
    if number of iterations $\geq \kappa$ then
        Maintain a list, $l$, of variables such that
          if one of them is flipped, $L(\mathbf{x}, \lambda)$ will improve.
        if $l$ is not empty then
          Update $\mathbf{x}$ by flipping the first element of $l$
        else if $\exists\ v$ s.t. $L(\mathbf{x}', \lambda) = L(\mathbf{x}, \lambda)$ when flipping $v$
          and number of consecutive flat moves $\leq L_f$
          and $v$ was not flipped in $L_t$ iterations then
          $\mathbf{x} \longleftarrow \mathbf{x}'$                    /* flat move */
        else
          Update $\lambda$: $\lambda \longleftarrow \lambda + c \cdot U(\mathbf{x})$
        end_if
    else
        if $\exists\ v$ s.t. $L(\mathbf{x}', \lambda) < L(\mathbf{x}, \lambda)$ when flipping $v$
          in a predefined order in $\mathbf{x}$ to get $\mathbf{x}'$ then
          $\mathbf{x} \longleftarrow \mathbf{x}'$
        else
          Update $\lambda$: $\lambda \longleftarrow \lambda + c \cdot U(\mathbf{x})$
        end_if
    end_if
    if iteration index mod $I_\lambda = 0$ then
        Reduce $\lambda$ for all clauses, e.g. $\lambda \longleftarrow \lfloor \lambda / r \rfloor$
    end_if
    if $\mathbf{x}$ is better than incumbent, keep $\mathbf{x}$ as incumbent.
end_while
```

Figure 2: $\mathcal{A}_3$, An efficient implementation of DLM.

compiled using *gcc* with the "-O" option and run on a 200-MHz Sun Ultra 1.

4.1 Optimal Reduction Interval I_λ

Dynamic reduction of Lagrange multipliers plays an improtont role in solving difficult SAT problems. Two related parameters in DLM, the reduction interval I_λ and the reduction ratio r, control the frequency and the magnitude of reduction of λ. In this section, we vary I_λ while keeping all the other parameters fixed in order to observe the relationship between the final result and I_λ.

We ran DLM 50 times from random initial points, each for 2 million iterations, for all 10 *"par"* problems. We set tabu length L_t be the same as flat-move limit L_f and tried clause weight w of 1, 2, and 4, flat-move limit L_f of 1, 5, 10, 30, 50, and 100, and reduction ratio r of 1.5 and 2. For each set of parameters, we varied the reduction interval I_λ from 1,000 to 500,000. Figure 3 shows some representitive results obtained by using parameter values $w = 2$ for all clauses, $L_t = 5$, $L_f = 5$, and $r = 1.5$. Here, we plot the average number of unsatisfied clauses in the solutions obtained by $\mathcal{A}_3$ against different I_λ on *"par16"* and *"par32."*

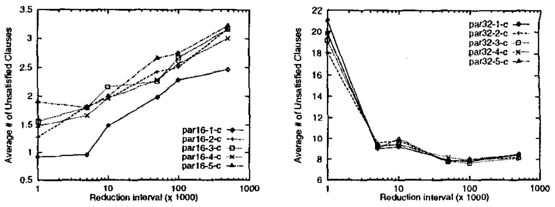


Figure 3: Average number of *unsatisfied* clauses in the solutions obtained by $\mathcal{A}_3$ with different reduction interval I_λ on *"par16"* and *"par32"* problems.

Figure 3 shows that a smaller I_λ is better for *"par16,"* while a larger I_λ is better for *"par32."* Our explanation is that the penalty for clause i represented in λ_i is determined based on the satisfiability status of the clause during the search and is increased if the clause is not satisfied. This is different from traditional Lagrangian methods for continuous problems that update λ continuously during the search. Here, DLM updates λ in a discrete manner only at local minima or in plateaus. These discrete increments of λ affect the balance of all λ's of a small SAT problem, such as *"par16,"* more than that of a large SAT problem, such as *"par32."* Hence, λ should be reduced more frequently for smaller problems in order to smooth out the gap due to discrete updates and reduce the imbalance among the λ's. As shown in Figure 3, I_λ = 1,000 or 5,000 works best for *"par16,"* and I_λ = 50,000 or 100,000 works best for *"par32."*

Next, we let DLM run until a solution was found for *"par16."* Using the same parameters as before, we tried I_λ of 1,000, 5,000, 10,000, and 50,000, and ran DLM 10 times from random initial points.

Table 1 shows the performance of DLM in solving *"par16"* with different I_λ's and compares it with the best results of GRASP [5], a greedy randomized adaptive search procedure that finds good quality solutions for a wide variety of combinatorial optimization problems. GRASP was run on an SGI Challenge with a 150 MHz MIPS R4400. We show the average CPU time of 10 random runs of GRASP. Both DLM and GRASP succeeded in all 10 runs for each problem. By comparing the best time of DLM to that of GRASP, DLM is about an order of magnitude faster than GRASP after considering the speed difference between Sun and SGI computers.

Table 1: Comparison of the execution time of $\mathcal{A}_3$ with different reduction interval I_λ with published results of GRASP on "par16" problems from the DIMACS archive [5].

	DLM $\mathcal{A}_3$ Sun Ultra 1 Seconds						GRASP
Problem	$I_\lambda = 1,000$			$I_\lambda = 5,000$	$I_\lambda = 10,000$	$I_\lambda = 50,000$	SGI Seconds
Identifier	Avg.	Min.	Max.	Avg.	Avg.	Avg.	Avg.
par16-1-c	203.8	1.1	889.9	153.4	263.9	407.3	2235.83
par16-2-c	370.1	2.1	1236.1	353.7	603.4	2546.7	2179.04
par16-3-c	338.5	52.1	1329.0	207.5	634.8	1013.5	2035.23
par16-4-c	321.0	7.6	742.6	242.7	418.7	1584.2	2071.30
par16-5-c	316.3	53.5	759.7	802.4	758.3	1967.3	2566.35

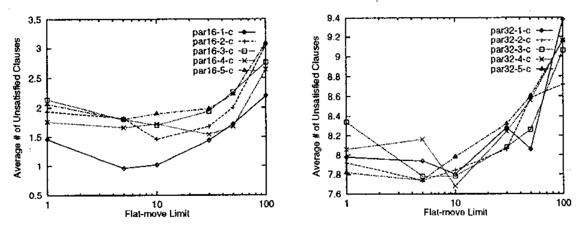


Figure 4: Average number of unsatisfied clauses in the solutions obtained by $\mathcal{A}_3$ with different flat-move limit L_f on "*par16*" and "*par32*" problems.

Table 2: Number of unsatisfied clauses in the final solutions when $\mathcal{A}_3$ was applied to solve "*par32*" problems.

	2 Million Iterations			20 Million Iterations		
Problem	$L_f = 5$		$L_f = 10$	$L_f = 5$		$L_f = 10$
Id.	Min.	Avg.	Min.	Min.	Avg.	Min.
par32-1-c	5	7.9	6	5	5.8	4
par32-2-c	5	7.7	5	3	5.6	4
par32-3-c	5	7.8	5	4	5.9	3
par32-4-c	5	8.2	3	4	5.8	3
par32-5-c	5	7.7	5	4	5.9	4

4.2 Optimal Flat-move Limit and Tabu Length

When DLM reaches a plateau, increasing the Lagrange multipliers immediately changes the surface of the plateau and may make it more difficult for the search to find a local minimum somewhere inside the plateau. To cope with this problem, our flat-move strategy allows the plateau to be searched without changing λ. On the other hand, it is not useful to search extensively an unpromising plateau. In DLM, the flat-move limit L_f is used to bound the time spent in a plateau. In this section, we vary L_f while fixing all other parameters in order to observe the relation between the final results obtained and different L_f.

Similar to the experiments before, we let the tabu length to be the same as the flat-move limit, and ran DLM 50 times from random initial points, each for 2 million iterations, for all problems. Figure 4 shows the result obtained by using the following parameter values: $w = 2$ for all clauses, $r = 1.5$, and $I_\lambda = 5,000$ for "*par16*," and $I_\lambda = 50,000$ for "*par32*." The average number of unsatisfied clauses in the solutions obtained by $\mathcal{A}_3$ is ploted against different flat-move limit L_f. Figure 4 shows that a small number of flat moves lead to the best performance, which is consistent with our analysis. Also, when the problem size increases from "*par16*" to "*par32*," the optimal flat-move limit is increased, but not by much.

Next, we ran DLM for more iterations on "*par32*." Using the good parameter values identified in our previous experiments, we extended the execution limit from 2 million iterations to 20 million iterations and ran DLM 50 times from random initial points. Table 2 shows the minumum and average number of unsatisfied clauses in the final solutions obtained by DLM when $I_\lambda = 50,000$, $r = 1.5$, and $L_f = L_t = 5$ and 10, respectively. In most cases, the number of unsatisfied clauses in the best solution was reduced by 1 or 2 when time was increased by 10 times, and the average number of unsatisfied clauses was also reduced by around 2. Overall, the best solutions that DLM found are assignments with 3 unsatisfied clauses for "*par32-2-c*," "*par32-3-c*," and "*par32-4-c*," and 4 unsatisfied clauses for "*par32-1-c*" and "*par32-5-c*."

4.3 Characteristics of Sub-Optimal Solutions

DLM formulates SAT as an optimization problem and searches for an optimal solution, which is also a feasible assignment. Sub-optimal solutions are those that have a few unsatisfied clauses. By examining the sub-optimal solutions found during a search, we can get some insights about the SAT problem and the search method.

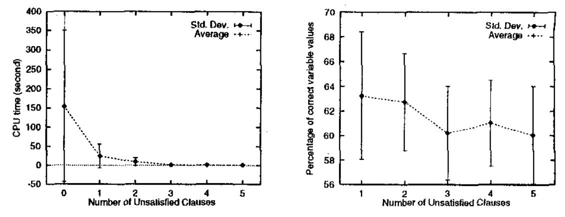
In general, DLM can find good sub-optimal solutions in very short time. However, its computation time increases exponentially when trying to find solutions with better quality. Figure 5 shows increases of computation time with respect to improvements of solution quality. We used the same parameter settings as in our previous experiments, and set $I_\lambda = 5,000$ and $L_f = 5$. We ran DLM 10 times from random initial points. In each run, we recorded the CPU time to reach sub-optimal solutions with a few unsatisfied clauses, in addition to the CPU time to reach the optimal solution. Figure 5a plots the average and standard deviation of CPU times used by DLM to find solutions with 0 to 5 unsatisfied clauses, respectively. We see that the CPU time increases significantly as the solution quality improves.

The time used by DLM to find a feasible solution varies significantly depending on the initial assignment. Table 1 shows that the maximum CPU time is often several orders more than the minimum CPU time in the 10 runs. Hence, the selection of good initial points in DLM is one way to improve its speed. Since DLM can find good sub-optimal solutions very quickly, one strategy for generating good initial points is to run DLM a number of times from random initial points and find a group of sub-optimal solutions. Methods similar to those used in genetic algorithms can then be used to combine these sub-optimal solutions to form initial points for DLM.

Unfortunately, our experiments show that this stragegy does not work well for hard SAT problems, such as "*par16*" and "*par32*." In these problems, optimal solutions have small attraction regions and are far away from sub-optimal solutions, and a good sub-optimal solution does not necessarily lead the search to an optimal solution.

To test this hypothesis experimentally, we found sub-optimal solutions for "*par16-1-c*" and computed the distance between them and the optimal solution (all "*par16*" problems have unique optimal solutions). Our results show that a good sub-optimal solution is usually not close to the optimal solution.

Figure 5b plots the average and standard deviation of percentage of variables in the sub-optimal assignments that have the same value as the optimal assignment. It shows that there are only about 63% of the variables on the average in the sub-optimal solutions that have the same values as in the optimal solution when the number of unsatisfied clauses is one. Further, the case with one unsatisfied clause (63%) is only slightly closer to the optimal solution than that with five unsatisfied clauses (60%).



(a) (b)

Figure 5: (a) Computational cost for finding sub-optimal solutions in problem "*par16-1-c*," and (b) similarity of the sub-optimal solutions to the optimal solution.

References

[1] M. Davis and H. Putnam. A computing procedure for quantification theory. *J. Assoc. Comput. Mach.*, 7:201–215, 1960.

[2] J. Gu. *Parallel Algorithms and Architectures for Very Fast AI Search*. PhD thesis, Dept. of Computer Science, University of Utah, August 1989.

[3] A. P. Kamath, N. K. Karmarkar, K. G. Ramakrishnan, and M. G. C. Resende. Computational experience with an interior point algorithm on the satisfiability problem. *Annals of Operations Research*, 25:43–58, 1990.

[4] D. G. Luenberger. *Linear and Nonlinear Programming*. Addison-Wesley Publishing Company, 1984.

[5] M. G. C. Resende and T. Feo. A GRASP for satisfiability. In D. S. Johnson and M. A. Trick, editors, *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge*, volume 26, pages 499–520. DIMACS Series on Discrete Mathematics and Theoretical Computer Science, American Mathematical Society, 1996.

[6] J. A. Robinson. A machine-oriented logic based on the resolution principle. *J. Assoc. Comput. Mach.*, pages 23–41, 1965.

[7] B. Selman and H. Kautz. Domain-independent extensions to GSAT: Solving large structured satisfiability problems. In *Proc. of the 13th Int'l Joint Conf. on Artificial Intelligence*, pages 290–295, 1993.

[8] B. Selman, H. Kautz, and D. McAllester. Ten challenges in propostional reasoning and search. In *Proc. 15th Int'l Joint Conf. on Artificial Intelligence*, pages 50–54. Morgan Kaufman, 1997.

[9] Y. Shang and B. W. Wah. A discrete Lagrangian-based global-search method for solving satisfiability problems. *J. of Global Optimization*, 12(1):61–99, January 1998.

[10] Zhe Wu. *Discrete Lagrangian Methods for Solving Nonlinear Distrete Constrained Optimization Problems*. M.Sc. Thesis, Dept. of Computer Science, Univ. of Illinois, Urbana, IL, May 1998.